PERGAMON

Transportation Research Part B 34 (2000) 241–253

TRANSPORTATION RESEARCH PART B

www.elsevier.com/locate/trb

Shortest paths in traffic-light networks

Yen-Liang Chen [a,*], Hsu-Hao Yang [b]

[a] *Department of Information Management, National Central University, Chung-Li 32054, Taiwan, ROC*
[b] *Department of Industrial Engineering and Management, National Chinyi Institute of Technology, Taiping 411, Taiwan, ROC*

Received 12 August 1998; accepted 22 April 1999

Abstract

The time-constrained shortest path problem (TCSPP) is an important generalization of the shortest path problem (SPP) and has attracted widespread research interest in recent years. This paper presents a novel time constraint, called traffic-light constraint, to simulate the operations of traffic-light control encountered in intersections of roads. Basically, the constraint consists of a repeated sequence of time windows. In each window, only the cars in specified routes are allowed to pass through the intersection. In a practical sense, this means that a car needs to wait if the light for its direction is red and can go if it is green. For this kind of network, a shortest path algorithm of time complexity $O(r \times n^3)$ is developed, where n denotes the number of nodes in the network and r the number of different windows in a node. In addition, we also prove that the time complexity of this algorithm is optimal. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Shortest path; Traffic light; Time window; Road network

1. Introduction

The shortest path problem (SPP) concentrates on finding the path with minimum distance, time, or cost from an origin to a destination through a connected network. It is a classical and important problem in the area of combinatorial optimization because of its numerous applications and generalizations in communications and transportation networks. Several excellent reviews on the SPP have been published, including Bodin et al. (1982), Deo and Pang (1984), and Golden and Magnanti (1977).

The time-constrained shortest path problem (TCSPP) is an important generalization of the SPP and has attracted much research interest over the past years. The basic notion is to consider when

* Corresponding author. Tel.: +886 3 4267266; fax: +886 3 425604.
E-mail address: ylchen@im.mgt.ncu.edu.tw (Y.L. Chen).

a node, under time constraints, in the network can be visited. The time window has been a common form of time constraints considered in the TCSPP, which requires that a node can be visited only in a specified time interval (e.g. Balakrishnan, 1993; Bramel and Simchilevi, 1996; Kolen et al., 1987; Russell, 1995). Restated, a time window defines the earliest time and the latest time that the node is available. In principle, two types of time windows are available. The first is the hard time window, where, if one or more time-window constraints are not satisfied, then the solution becomes infeasible (e.g. Bramel and Simchilevi, 1996; Kolen et al., 1987; Russell, 1995). The second is the soft time window, where a cost penalty is incurred if the node's arrival is outside its time window. The penalty can be assumed to be a linear function of the amount of violation (Balakrishnan, 1993). When the hard time window is considered, a common objective is to find the least-cost path from the source node s to the destination node d such that all intermediate nodes are visited within their respective time windows. When considering the soft time window, minimization of total cost is also a common objective. However, the intermediate nodes may be visited outside their corresponding time windows and the penalty associated with the time window violation is an additional cost component in the total cost.

This paper is interested in finding the shortest paths in a modern city that has traffic-light controls in a number of intersections of roads. Let a network $N = (V, A, t, s, d)$ denote a city, where the node set V corresponds to the intersections and the arc set A the roads in the city, $t(u, v)$ the time from node u to node v, s the source node and d the destination node. Then, our goal is to find a shortest path from node s to node d in N, where some nodes are constrained to the traffic-light controls. An immediate problem encountered is how to model the traffic-light control. Traditional soft time window appears to be a promising alternative because it designates at what time period we can pass through the node and otherwise. However, two properties cannot be completely described by the traditional soft time window in light controls. The first one is that the light control in an intersection usually contains a repeated sequence of time windows with designated durations. As a result, it is imperative that the soft time window should be extended from a single window to a repeated sequence of multiple windows. The second property is that the traditional soft time window allows one to pass through the node if its arrival falls into the range of time windows. In practical light control, however, each window may allow only some route passages. Consider an example in Fig. 1(a) that depicts an intersection of two roads. Assume the intersection has a light control consisting of a repeated sequence of four different windows. In the first window, only the routes indicated in Fig. 1(b) are allowed. In the second window, some other routes indicated in Fig. 1(c) are eligible to pass. The third and fourth windows in essence resemble the first and second ones but slightly differ in directions: the former is between north and south, while the latter is between east and west.

From the above discussions, we assume that $V = V_1 \cup V_2$ and $\{s, d\} \in V_1$, where V_1 denotes the node set without window constraints and V_2 represents the node set with window constraints. For each node $u \in V_2$, it is associated with a window-list $WL(u) = (ws_u, w_{u,1}, w_{u,2} \ldots, w_{u,r})$, where ws_u is the starting time of the first window and $w_{u,i}$ the ith time window of node u for $i = 1, \ldots, r$. Notably, since these windows form a repeated sequence, by assuming $w_{u,0} = w_{u,r}$, we have the relationship that $w_{u,(k\times r)+i} = w_{u,i}$ for any nonnegative integers k and i, where $i \leqslant r$. For each window $w_{u,i}$ of node u, we use $d_{u,i}$ to describe how long this window lasts. Besides, we also associate a set of node-triplets $NT_{u,i}$ with each window $w_{u,i}$. A node-triplet $\langle x, u, y\rangle$ in $NT_{u,i}$ implies that the ith window of node u allows one to visit node y through node x. In other words, $NT_{u,i}$

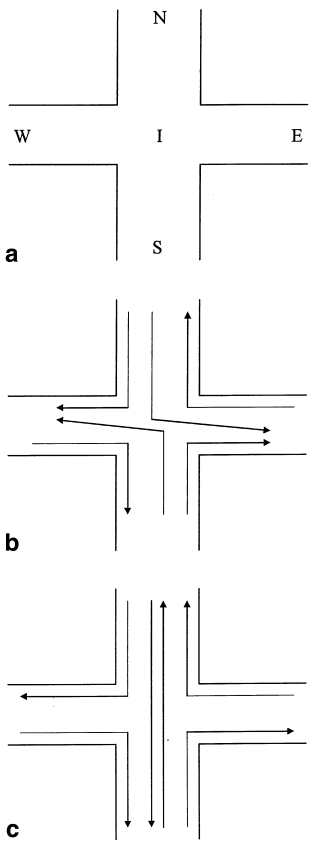

Fig. 1. (a) A sample intersection of roads. (b) The light control in the first window. (c) The light control in the second window.

denotes the set of allowable routes in the *i*th time window of node *u*. As an illustration, consider Fig. 2 which is the network representation of Fig. 1(a). To node *I*, we attach four time windows $w_{I,1}$, $w_{I,2}$, $w_{I,3}$ and $w_{I,4}$. In so doing, window $w_{I,1}$ has a set of node-triplets

$$NT_{I,1} = \{\langle N,I,W\rangle,\ \langle N,I,E\rangle,\ \langle S,I,W\rangle,\ \langle S,I,E\rangle,\ \langle W,I,S\rangle,\ \langle E,I,N\rangle\}.$$

In a similar way,

$$NT_{I,2} = \{\langle N,I,W\rangle,\ \langle N,I,S\rangle,\ \langle S,I,N\rangle,\ \langle S,I,E\rangle,\ \langle W,I,S\rangle,\ \langle E,I,N\rangle\},$$

$$NT_{I,3} = \{\langle W,I,S\rangle,\ \langle W,I,N\rangle,\ \langle E,I,S\rangle,\ \langle E,I,N\rangle,\ \langle S,I,E\rangle,\ \langle N,I,W\rangle\},$$

and

$$NT_{I,4} = \{\langle W,I,S\rangle,\ \langle W,I,E\rangle,\ \langle E,I,W\rangle,\ \langle E,I,N\rangle, \langle S,I,E\rangle,\ \langle N,I,W\rangle\}.$$

By this example, we assume that "right turn on red" is allowed. Note that the node-triplet set $NT_{I,1}$ contains all the routes in Fig. 1(b), and $NT_{I,2}$ all the routes in Fig. 1(c).

The traffic-light network has a greater expressive power to model the modern city with light controls since we can represent its traffic-light constraints in a more appropriate way. The rationale is as follows. For an intersection with entering and leaving lanes, the light control contains

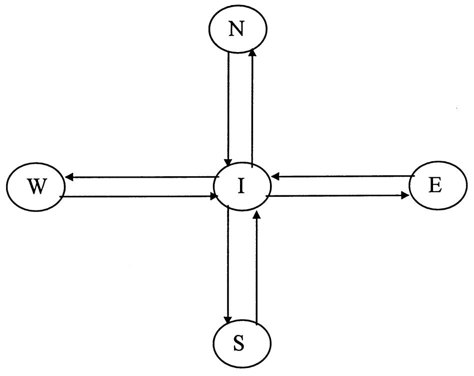

Fig. 2. The network representation of Fig. 1(a).

a repeated sequence of windows, where each window allows only some routes and prohibits others. Let each allowable route be decomposed into an entering arc, say (x, u), and a leaving arc (u, y). Then, all allowable routes (x, u, y), if denoted by $\langle x, u, y\rangle$, are contained in the node-triplet set of the corresponding time window. In so doing, the traffic-light network model of the city is constructed. The problem of how to find the quickest path to go through a city with many traffic-light controls reduces to solve the SPP in the constructed traffic-light network.

The content of this paper is organized as follows. In Section 2, we develop a solution method to find the shortest path in this network. Section 3 includes conclusions, limitations and future research directions.

2. The solution algorithm

Let $N = (V_1 \cup V_2, A, WL, t, s, d)$ denote a traffic-light network, where V_1 denotes the node set without window constraints, V_2 represents the node set with window constraints, A is the arc set without multiple arcs and self-loops, $t(u, v)$ the length of time of arc $(u, v) \in A$. For each node $u \in V_2$, it is associated with a window-list $WL(u) = (ws_u, w_{u,1}, w_{u,2}, \ldots, w_{u,r})$, where ws_u is the starting time of the first window and $w_{u,i}$ the ith time window of node u for $i = 1, \ldots, r$. Besides, each window $w_{u,i}$ is associated with a duration $d_{u,i}$ and a set of node-triplets $NT_{u,i}$, where a node-triplet $\langle x, u, y\rangle$ is in $NT_{u,i}$ if visiting node y through node x is allowed in window $w_{u,i}$. Since we represent windows using a repeated sequence (recall the earlier relationship $w_{u,(k\times r)+i} = w_{u,i}$), it is the very nature to contain multiple cycles and some way is required to treat arrivals in later cycles of the signals. To resolve this matter, we choose to use the function "mod" and will be shown in the example later. Also note that we assume the sequence of the windows describes the whole phasing of the signal.

Since a node u in V_1 can be regarded as a node in V_2 by associating it with a window of infinite duration and containing all possible node-triplets, we therefore assume that all the nodes are in set V_2 for ease of presentation.

Before presenting the algorithm, we introduce the following labels and function.

- For each arc (v,u) in A, let $arrived(v,u)$ denote the earliest time to arrive at node u through arc (v,u).
- For a node $u \in V_2$, let $leaving(v,u,w)$ denote the earliest time to leave node u if the preceding arc is (v,u) and the succeeding arc to travel is (u,w).
- Let $P^*(u,w)$ denote the shortest path to node w through arc (u,w).
- $pred(u,w) = (v,u)$ means that the second-to-the-last arc in path $P^*(u,w)$ is the arc (v,u).
- The function $earliest(v,u,w,t)$ computes the earliest time to leave node u, provided that we visit node u through arc (v,u) at time t and the next arc to travel is (u,w).

Notice that if we visit node u from v at time $arrived(v,u)$, we may not be able to leave for node w immediately. Chances are we need to wait until the coming window including the node-triplet $\langle v,u,w\rangle$. Therefore, we have the following relation

$$leaving(v,u,w) = earliest(v,u,w,arrived(v,u)).$$

After leaving node u, we will visit node w at the time $leaving(v,u,w) + t(u,w)$. Since we may visit node u through a variety of arcs (v,u), we have

$$arrived(u,w) = \min_{\text{for all } v} \{leaving(v,u,w) + t(u,w)\}.$$

Based on the above discussion, we give the following algorithm to find the shortest path in the underlying network.

Algorithm minimum-time

```
1. Set $arrived(0,s) = 0$.
      Set all $arrived(v,u) = \infty$ for all arcs $(v,u)$ in $A$.
      Insert all values of $arrived(v,u)$ into the set $HP$.
2. Find and remove the minimum element $arrived(v,u)$ from $HP$.
3. If $u = d$ then go to step 5.
4. For each arc $(u,w)$ emanating from node $u$, do
         Begin
            $leaving(v,u,w) = earliest(v,u,w,arrived(v,u))$.
            $temp(u,w) = leaving(v,u,w) + t(u,w)$.
            If $temp(u,w) < arrived(u,w)$ then
             $arrived(u,w) = temp(u,w), pred(u,w) = (v,u)$, and
             update the value $arrived(u,w)$ in $HP$.
         End.
      Go to step 2.
5. From $pred(v,u)$ we find the shortest path.
   Output $arrived(v,u)$ as the minimum time.
```

Example 1. Consider the traffic-light network shown in Fig. 3, where the number along each arc is the arc's time. Beside each node, we also show its window restriction. For example, the first window of node C starts at time 3; window $w_{C,1+2i}$ has a duration of 2 time units and the duration of window $w_{C,2+2i}$ is 4 time units where i is a nonnegative integer. The triplet $\langle A,C,d\rangle$ is the allowable route in window $w_{C,1+2i}$, while $\langle B,C,d\rangle$ and $\langle D,C,d\rangle$ are allowable in window $w_{C,2+2i}$.

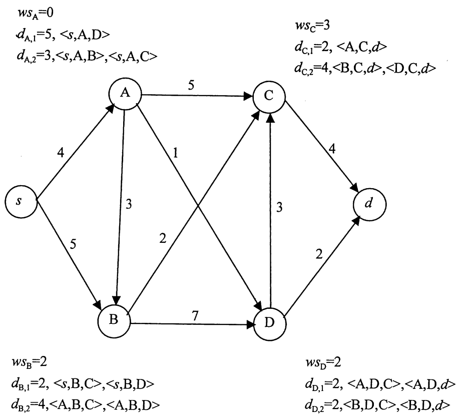

Fig. 3. The original network with traffic-light controls.

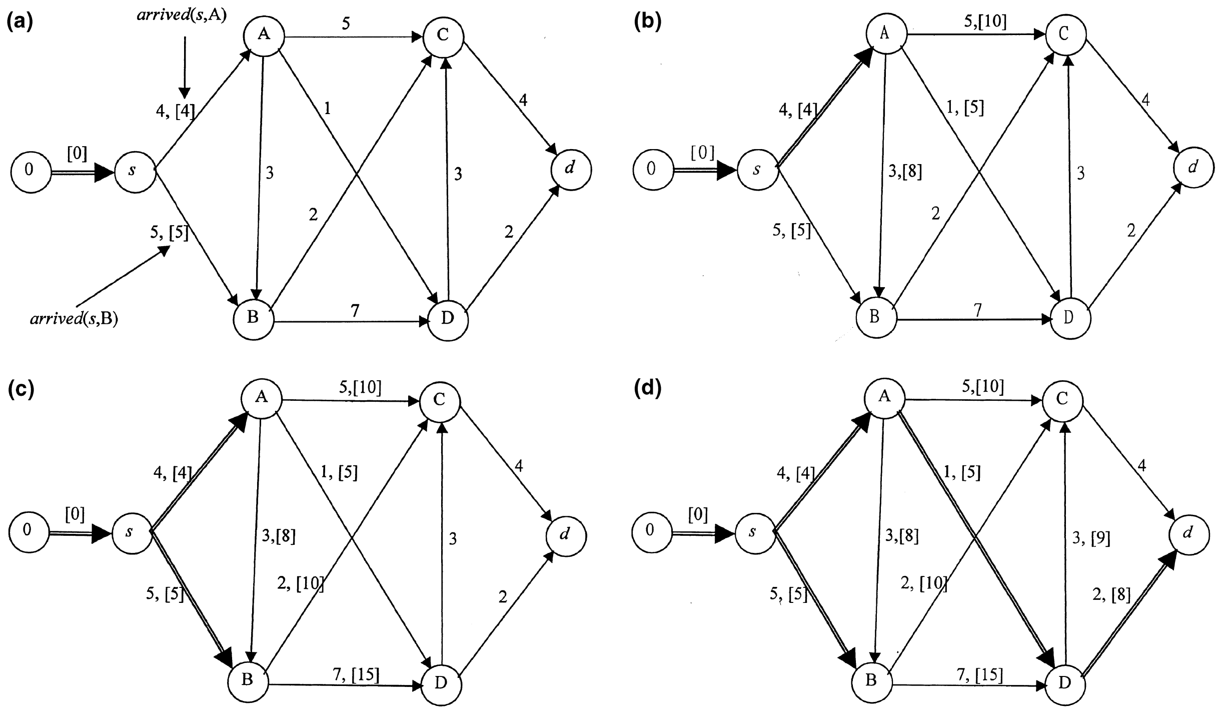

Fig. 4. (a) The first iteration of Algorithm minimum-time. (b) The second iteration of Algorithm minimum-time. (c) The third iteration of Algorithm minimum-time. (d) The fourth and last iterations of Algorithm minimum-time.

The execution of the algorithm is shown in Fig. 4, where the number in square brackets along an arc denotes the value of $arrived(v, u)$. Initially, we choose $arrived(0, s)$ from *HP*. Since there are two arcs leaving node *s*, i.e., arcs (s, A) and (s, B), we need to compute the values of *arrived*(*s*,A) and *arrived*(*s*,B). The result of the first iteration is shown in Fig. 4(a). In the second iteration, the minimum value in *HP* is $arrived(s, \mathrm{A}) = 4$. Although we visit node A at time 4, we are not allowed to leave for nodes B or C until window $w_{\mathrm{A},2}$. That explains why $leaving(s, \mathrm{A}, \mathrm{B}) = 5$, $leaving(s, \mathrm{A}, \mathrm{C}) = 5$, $arrived(\mathrm{A}, \mathrm{B}) = 8$ and $arrived(\mathrm{A}, \mathrm{C}) = 10$. On the contrary, we can leave for node D immediately, and hence $leaving(s, \mathrm{A}, \mathrm{D}) = 4$ and $arrived(\mathrm{A}, \mathrm{D}) = 5$. Fig. 4(b) shows the second iteration. The third iteration as shown in Fig. 4(c) chooses $arrived(s, \mathrm{B}) = 5$ from *HP*. Since window $w_{\mathrm{B},1}$ contains the node-triplets $\langle s, \mathrm{B}, \mathrm{C}\rangle$ and $\langle s, \mathrm{B}, \mathrm{D}\rangle$, we must wait until the coming window is $w_{\mathrm{B},1}$. Therefore, the earliest leaving time from node B is time 8, and $arrived(\mathrm{B}, \mathrm{C}) = 10$ as well as $arrived(\mathrm{B}, \mathrm{D}) = 15$. After two more iterations, the minimum element chosen from *HP* is $arrived(\mathrm{D}, d)$, which satisfies the terminating condition. From Fig. 4(d), we see that the path is $(s, \mathrm{A}, \mathrm{D}, d)$ and the minimum time to arrive at node d is 8 time units.

The following theorem shows the validity of the algorithm.

Theorem 1. *Algorithm minimum-time is correct.*

Proof. We prove by induction. At any iteration, the algorithm partitions all arcs into two sets, namely, *HP* and $\overline{HP}$, where $\overline{HP} = A - HP$. Our induction hypotheses are founded on the premises that: (1) the time label $arrived(v, u)$ of each arc (v, u) in $\overline{HP}$ is optimal, and (2) the time label $arrived(v, u)$ of each arc (v, u) in *HP* is the total time of the shortest path from s to u through (v, u) provided that each intermediate arc in the path lies in $\overline{HP}$. We perform induction according to the cardinality of the set $\overline{HP}$.

To prove hypothesis (1), recall that at each iteration the algorithm moves an arc (v, u) in the set *HP* with the smallest value to the set $\overline{HP}$. It leaves to show that $arrived(v, u)$ of arc (v,u) is optimum. Notice that by our induction hypotheses, $arrived(v, u)$ is the total time of a shortest path to node u through arc (v, u) among all paths that does not contain any intermediate arc in *HP*. We now show that the total time of any path from s to u through arc (v, u) that contains some arcs in *HP* as an intermediate arc will be at least $arrived(v, u)$. To do that, consider any path P from the source to node u through (v, u) that contains at least one arc in *HP* as an intermediate arc. The path P can be decomposed into two segments P_1 and P_2, where P_1 does not contain any arc in *HP* as an intermediate arc but the last arc, say (h, k), is in *HP*. By the induction hypotheses, this suggests that the total time of P_1 is at least $arrived(h, k)$. Moreover, since arc (v, u) is the smallest time label in *HP*, $arrived(h, k) \geqslant arrived(v, u)$. Therefore, the path segment P_1 has total time at least $arrived(v, u)$. Furthermore, since all arc times are nonnegative, the total time of the path segment P_2 is nonnegative. Consequently, the total time of path P is no less than $arrived(v, u)$. This result establishes the fact that $arrived(v, u)$ is the shortest path total time of node u through (v, u) from the source node.

We next show that the algorithm preserves the hypothesis (2). After the algorithm has labeled a new arc (v, u) permanently, the time labels of some arcs in $HP - \{(v, u)\}$ may decrease since arc (v, u) could become an intermediate arc in the temporary shortest paths to these arcs. But recall that after permanently labeling arc (v, u), the algorithm examines each arc (u, w) emanating from node u and sets

$$arrived(u, w) = leaving(v, u, w) + t(u, w)$$

if

$$leaving(v, u, w) + t(u, w) < arrived(u, w).$$

Therefore, after the time label update operation, the time label of each arc (u, w) in $HP - \{(v, u)\}$ is the total time of a shortest path from node s to arc (u, w) subject to the restriction that each intermediate arc in the path must belong to $\overline{HP} \cup (v, u)$. □

Having established the correctness of the algorithm, we will address two issues that eventually lay the foundation for determining the time complexity of the algorithm. These issues are: (1) How to compute the function *earliest*(v, u, w, t) in step 4 of Algorithm minimum-time? (2) What is the underlying data structure to implement the set HP and what is the time complexity of Algorithm minimum-time?

2.1. The computation of the function earliest(v, u, w, t)

Suppose that node u has r windows $w_{u,1}, w_{u,2}, \ldots, w_{u,r}$. Let n denote the total number of nodes in V. Then each window has at most n^2 node-triplets $\langle v, u, w\rangle$, since v and w are in $V - \{u\}$. Therefore, if we find the earliest leaving time by sequentially searching all the node-triplets in the windows of node u, we need time at least $O(rn^2)$ for each computation of *earliest*(v, u, w, t). This may to some extent affect the performance of our algorithm. To overcome this drawback, we deploy a strategy to preprocess every node in the network before we run the algorithm. After processing node u, we get an $n \times n$ matrix M_u, where $M_u(v, w)$ records what windows contain the node-triplet $\langle v, u, w\rangle$. Since each entry of $M_u(v, w)$ contains no more than r windows, it is expected that each call of the function *earliest*(v, u, w, t) can be answered very quickly.

2.1.1. Preprocessing every node u in the network

To build the matrix M_u, we need to scan the associated windows of node u. Initially, we set all entries of $M_u(v, w) = \emptyset$, then scan from window $w_{u,1}$ to window $w_{u,r}$. While scanning window $w_{u,i}$, if we find that a node-triplet, say $\langle v, u, w\rangle$, appears in the window, we then insert $w_{u,i}$ into the entry $M_u(v, w)$. After finishing the processing, a list of windows is attached with each entry $M_u(v, w)$. Obviously, since there are at most n^2 node-triplets $\langle v, u, w\rangle$ in each window, this processing for each node can be done in time $O(rn^2)$, and the entire network $O(rn^3)$.

For ease of presentation, we initiate a function *on–off* $(w_{u,i})$ to denote if window $w_{u,i}$ is present in the entry $M_u(v, w)$. In other words, if window $w_{u,i}$ appears in the entry $M_u(v, w)$, we set *on–off*$(w_{u,i}) =$ on; otherwise, we set *on–off* $(w_{u,i}) =$ off. According to this scheme, each entry $M_u(v, w)$ consists of a list of r on–off switches, where the ith switch indicates that whether window $w_{u,i}$ is present in the entry $M_u(v, w)$.

With the design of on–off switches, let us take a look at how they can facilitate the execution. Suppose we visit node u at time t and t is in window $w_{u,i}$. If *on–off* $(w_{u,i}) =$ on in entry $M_u(v, w)$, then *earliest*$(v, u, w, t) = t$ because we can leave immediately. On the other hand, if *on–off*$(w_{u,i}) =$ off, we must wait until the next window $w_{u,j}$ satisfying *on–off* $(w_{u,j}) =$ on. By using an attribute called *next-window* $(w_{u,i})$, we give the relationship as below.

$$
\begin{aligned}
&\textit{next-window}(w_{u,i}) = w_{u,i} && \text{if } \textit{on–off}(w_{u,i}) = \text{on in } M_u(v, w),\\
&\textit{next-window}(w_{u,i}) = w_{u,j} && \text{if } \textit{on–off}(w_{u,i}) = \text{off in } M_u(v, w) \text{ and } w_{u,j} \text{ is the next window}\\
& && \text{satisfying } \textit{on–off}(w_{u,j}) = \text{on}
\end{aligned}
$$

These attributes can be easily constructed in O(r) time by scanning the on–off switch list of entry $M_u(v, w)$ one time. The procedure is given in below.

```
1. Find the first window $w_{u,i}$ satisfying on–off ($w_{u,i}$) = on.
   Set $next = w_{u,i}$.
2. For $i = r$ to 1.
       If on–off ($w_{u,i}$) = on, then next-window($w_{u,i}$) = $w_{u,i}$
                                        next = $w_{u,i}$
       Else next-window ($w_{u,i}$)) = next.
```

Since the above procedure is executed for all n^2 entries in matrix M_u, the total time for computing matrix M_u is O(rn^2), and the whole network O(rn^3).

In summary, the preprocessing stage contains two major parts, i.e., to get on–off switch lists and to compute the *next-window* attributes. Totally, they can be done in O(rn^3) for the whole network. The following is an example to illustrate the preprocessing works.

Example 2. Let us consider a sample network with four nodes A, B, C and D and what we concern here is node D. Assume that there are five windows associated with node D as follows.

$$
\begin{array}{ll}
w_{D,1} & \{\langle B, D, C\rangle, \langle C, D, A\rangle\}, \\
w_{D,2} & \{\langle C, D, A\rangle, \langle A, D, C\rangle, \langle B, D, A\rangle\}, \\
w_{D,3} & \{\langle A, D, B\rangle, \langle C, D, C\rangle, \langle C, D, B\rangle\}, \\
w_{D,4} & \{\langle B, D, A\rangle, \langle B, D, C\rangle, \langle A, D, C\rangle\}, \\
w_{D,5} & \{\langle A, D, B\rangle, \langle B, D, A\rangle\}.
\end{array}
$$

By scanning the windows of node D sequentially, we can build the initial matrix M_D as shown in Table 1. The matrix can be easily represented in a form of on–off switch list. For example, the on–off switch lists of entries $M_D(B, C)$ and $M_D(A, B)$ are shown in Table 2. Finally, we need to compute the *next-window* attributes, and the results concerning $M_D(B, C)$ and $M_D(A, B)$ are also shown in Table 2.

Now we present the following procedure to find the earliest leaving time.

2.1.2. Finding the earliest leaving time for a given t

Having obtained attributes *next-window*($w_{u,i}$) for entry $M_u(v, w)$, the function *earliest*(v, u, w, t) can be determined as follows.

1. Determine the window containing the value t. Without loss of generality, assume that t is in window $w_{u,i}$.
2. If *next-window* ($w_{u,i}$) = $w_{u,i}$ then *earliest*(v, u, w, t) = t Else *earliest*(v, u, w, t) =the beginning time of *next-window* ($w_{u,i}$).

Table 1
The matrix M_D

	A	B	C
A	$\emptyset$	$w_{D,3}$, $w_{D,5}$	$w_{D,2}$, $w_{D,4}$
B	$w_{D,2}$, $w_{D,4}$, $w_{D,5}$	$\emptyset$	$w_{D,1}$, $w_{D,4}$
C	$w_{D,1}$, $w_{D,2}$	$w_{D,3}$	$w_{D,3}$

Table 2

	On–off switch	Next-window
$M_D(B, C)$		
$w_{D,1}$	On	$w_{D,1}$
$w_{D,2}$	Off	$w_{D,4}$
$w_{D,3}$	Off	$w_{D,4}$
$w_{D,4}$	On	$w_{D,4}$
$w_{D,5}$	Off	$w_{D,1}$
$M_D(A, B)$		
$w_{D,1}$	Off	$w_{D,3}$
$w_{D,2}$	Off	$w_{D,3}$
$w_{D,3}$	On	$w_{D,3}$
$w_{D,4}$	Off	$w_{D,5}$
$w_{D,5}$	On	$w_{D,5}$

In step 1 of the above procedure, we are to find the window $w_{u,i}$ containing t for the case of multiple cycles. Given a t, this can be achieved by the following.
1. Set $a = \lfloor (t - ws_u)/TD_u \rfloor$.
2. Set $b = (t - ws_u) \bmod TD_u$.
3. Find a value i such that $D_{u,i-1} \leqslant b < D_{u,i}$.
4. Output window $w_{u,i}$.
where $D_{u,i} = \sum_{k=1}^{i} d_{u,k}$ and $TD_u = \sum_{k=1}^{r} d_{u,k}$. Since $D_{u,i}$ and TD_u can be computed beforehand, these values can be used directly. Thus, the above procedure can be done in time O(log r), since the bottleneck lies in searching the value of i satisfying $D_{u,i-1} \leqslant b < D_{u,i}$ that can be done in time O(log r) by binary search. Therefore, we have the following lemma.

Lemma 1. *After the preprocessing of the network requiring time* O(rn^3), *each call of the function earliest*(v, u, w, t) *can be answered in time* O(log r).

Example 3. Suppose node u has seven windows with $d_{u,1} = 2$, $d_{u,2} = 3$, $d_{u,3} = 5$, $d_{u,4} = 2$, $d_{u,5} = 1$, $d_{u,6} = 3$, $d_{u,7} = 5$ and the window starting time $ws_u = 4$. From these durations, we have $D_{u,0} = 0$, $D_{u,1} = 2$, $D_{u,2} = 5$, $D_{u,3} = 10$, $D_{u,4} = 12$, $D_{u,5} = 13$, $D_{u,6} = 16$, $D_{u,7} = 21$ and $TD_u = 21$. Suppose, among these seven windows, windows $w_{u,2}$, $w_{u,5}$ and $w_{u,6}$ contain the node-triplet $\langle v, u, w \rangle$. By scanning these windows from $w_{u,1}$ to $w_{u,7}$, we construct the on–off switch list of entry $M_u(v, w)$ as shown in the first column of Table 3. From this on–off switch list, we further construct the *next-window* attributes as shown in the second column. Now, given a $t = 87$, we get $a = \lfloor (87 - 4)/21 \rfloor = 3$, $b = (87 - 4) \bmod 21 = 20$. By searching among the values $D_{u,1}, D_{u,2}, \ldots, D_{u,7}$, we find that the window containing $b = 20$ is window $w_{u,7}$. Since *next-window* $(w_{u,7}) = w_{u,2}$, we have the earliest leaving time, i.e., the beginning time of the next window $w_{u,2}$, as

$$ws_u + (TD_u \times (a + 1)) + D_{u,2-1} = 4 + (21 \times 4) + 2 = 90.$$

If given a different $t = 94$, then we have $a = \lfloor (94 - 4)/21 \rfloor = 4$, $b = (94 - 4) \bmod 21 = 6$. By searching among the values $D_{u,1}, D_{u,2}, \ldots, D_{u,7}$, we find that the window containing $b = 6$ is window

Table 3
The structure of entry $M_u(v, w)$

	On–off switch	Next-window
$w_{u,1}$	Off	$w_{u,2}$
$w_{u,2}$	On	$w_{u,2}$
$w_{u,3}$	Off	$w_{u,5}$
$w_{u,4}$	Off	$w_{u,5}$
$w_{u,5}$	On	$w_{u,5}$
$w_{u,6}$	On	$w_{u,6}$
$w_{u,7}$	Off	$w_{u,2}$

$w_{u,3}$. Since *next-window* $w_{u,3}) = w_{u,5}$, we have the earliest leaving time, i.e., the beginning time of the next window $w_{u,5}$, as

$$ws_u + (TD_u \times a) + D_{u,5-1} = 4 + (21 \times 4) + 12 = 100.$$

2.2. Data structure for HP and the time complexity

In efforts to obtain the time complexity of the algorithm Minimum-time, recall that the set *HP* involves operations of insertion (step 1), find-minimum and delete-minimum (step 2), decrease-value (step 4) and calling the function *earliest* (step 4). In light of these facts, consider using Fibonacci heap to implement the set *HP* (Fredman and Tarjan, 1987). The advantage of using Fibonacci heap is that insertion, decrease-value and find-minimum operations can all be done in O(1) amortized time, and delete-minimum operation in amortized time O(log h), where h is the heap size. Then we have the following lemma.

Lemma 2. *If we use the Fibonacci heap data structure to store arrived*(v, u) *for every* (v, u) *in A, then the time complexity of Algorithm minimum-time is* $O(mn \log r + rn^3) = O(r \times n^3)$, *where n is the number of nodes, m the number of arcs and r the number of different windows in a node.*

Proof. Since *HP* contains at most m arcs, it is obvious that the total time for step 1 (insertion) is $O(m)$, step 2 (find-minimum and delete-minimum) is $O(m \log m)$. Two operations, namely, decrease-value and calling the function *earliest*, jointly determine the complexity of step 4. As far as decrease-value is concerned, we need to examine n nodes each with at most m arcs to update the value *arrived*(u, w). Therefore, the total number for this part is simply $O(mn)$. Next, in each examination of arc (u, w), we need to compute its leaving time by calling the function *earliest*$(v, u, w, arrived(v, u))$. Since there are $O(mn)$ examinations and each call can be done in $O(\log r)$ time as shown in Lemma 1, the total time for calling functions is $O(mn \log r)$. That is, total time for step 4 is $O(mn \log r)$. Adding up the preprocessing time, which is $O(r \times n^3)$ in Lemma 1, the time complexity is $O(mn \log r + r\,n^3) = O(r \times n^3)$. □

Finally, because of Lemma 2, the following suffices to show our algorithm has an optimum time complexity.

Lemma 3. *The time complexity of Algorithm minimum-time is optimal.*

Proof. The network has n nodes, where each node has at most r windows. Besides, there are at most n^2 node-triplets in each window. Taking account of these facts, the number of input parameters of the problem may have a size up to $r \times n^3$. If we are to solve a problem, each input parameter of the problem must be examined at least once. That means the lower bound of the complexity to solve the problem is $O(r \times n^3)$. By Lemma 2, we therefore know that $O(r \times n^3)$ is the optimum time complexity. □

3. Conclusions

This paper has two major contributions. First, our traffic-light model shows to be a powerful tool in solving various kinds of path problems encountered in a modern city with light controls in many intersections. The great strength of the model attributes to the traffic-light constraint that demonstrates the ability to simulate the operations of a practical light control. By incorporating such a constraint into a node, when and what routes one is allowed to pass through the intersection can be easily specified. The second contribution is that a polynomial algorithm is developed for finding the minimal total time path. Besides a polynomial time complexity, we further show that the complexity is optimal.

Several limitations should be noted in this paper. First, no inter-green times are used in the model. In reality between the end of phase i and the start of green in phase $i + 1$ there is an inter-green time due to safety reasons. Second, the proposed algorithm does not take into account limited capacity during green nor queuing effects at the stop-line. In the model it is assumed that all the waiting vehicles can immediately start regardless of the number of queued vehicles. In the real world problems, the number of vehicles being able to pass the signal during green depends on the number of lanes. This limitation underestimates delays that may be significant with traffic demand approaching capacity and in over-saturated conditions.

Finally, we briefly mention possible extensions of the paper. Since the total time of a path consists of the travelling time and the waiting time, a natural extension is to consider the problem under a combination of the goals such as the total time, the travelling time and the waiting time. Moreover, we can consider the vehicle routing problem in a traffic-light network rather than the traditional network. For example, we can assume that some nodes are required to be visited; some node is the depot and a vehicle route should not spend time more than a given threshold. The problem becomes how to send the minimum number of vehicles with a given capacity to complete the service in a traffic-light network. In a similar manner, a number of vehicle scheduling problems can be introduced in a traffic-light network.

Acknowledgements

The authors are grateful to the anonymous referees for their helpful comments.

References

Balakrishnan, N., 1993. Simple heuristics for the vehicle routing problem with soft time windows. Journal of Operational Research Society 44, 279–287.

Bodin, L.D., Golden, B.L., Assad, A.A., Ball, M.O., 1982. Routing and scheduling of vehicles and crews: the state of the art. Computers & Operations Research 10, 63–211.

Bramel, J., Simchilevi, D., 1996. Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. Operations Research 44, 501–509.

Deo, N., Pang, C., 1984. Shortest path algorithms: taxonomy and annotation. Networks 14, 275–323.

Fredman, M.L., Tarjan, R.E., 1987. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of ACM 34, 596–615.

Golden, B.L., Magnanti, T.L., 1977. Deterministic network optimization: a bibliography. Networks 7, 149–183.

Kolen, A.W.J., Rinnooy Kan, A.H.G., Trienekens, H.W.J.M., 1987. Vehicle routing with time windows. Operations Research 35, 266–273.

Russell, R.A., 1995. Hybrid heuristics for the vehicle-routing problem with time windows. Transportation Sciences 29, 156–166.